

REMARKS

At the outset, the Examiner is thanked for the courtesy extended during the telephone interview conducted September 28, 2004. During that interview, the following issues were discussed.

With regard to the rejection under 35 U.S.C. § 112, second paragraph, as applied to the independent claims (Office Action, ¶ 8A), the Examiner stated that those claims did not recite that steps (a) and (b) include two-dimensional scans having at least one corresponding dimension. Proposed claim language to overcome the rejection was discussed. With regard to the rejection under 35 U.S.C. § 112, second paragraph, specifically applied to claims 6-9 and 33-36 (Office Action, ¶ 8B), the Examiner stated that the general recitation of a hill-climbing technique did not specify that a maximum was reached and could be broadly read on several techniques. The Examiner also requested the citation of a reference teaching hill climbing. Finally, with regard to the grounds of rejection involving *Higashi*, the English translation of that reference was discussed, as was the following difference between the applied reference and the present invention. The applied reference teaches a projection technique, namely, MIP. By contrast, the present invention solves for actual values in image space of the unknown high-resolution voxel values to form the isotropic, high-resolution, three-dimensional image in the image space. Again, proposed claim language to overcome the rejection was discussed.

The Applicants hereby respond to the Office Action mailed May 18, 2004, as follows.

In response to the objection to the drawings, a Submission of Formal Drawings is filed concurrently herewith.

In response to the rejection of claims 1, 2, 4-10, 12-13, 28, 29, 31-37 and 39-40 under 35 U.S.C. § 112, second paragraph, the Applicants respectfully submit that the present Amendment

overcomes that rejection. In addition, the Applicants attach hereto a copy of the relevant portion of Rich et al, *Artificial Intelligence, Second Edition*, © 1991 McGraw-Hill, Inc., which explains hill climbing.

The Applicants further submit that the invention defined by present claims 1, 2, 4, 5, 10, 12, 13, 28, 29, 31, 32, 37, 39 and 40 would not have been obvious over *Higashi*.

The present claimed invention is not directed merely to a projection technique. Instead, the present claimed invention solves for actual values of the unknown high-resolution voxel values in the image space.

To that end, the present claimed invention has the following noteworthy characteristics. The image data of the first plurality of slices are two-dimensional image data. The image data of the second plurality of slices are also two-dimensional image data and have at least one dimension substantially in common with the image data of the first plurality of slices. The first and second plurality of slices are registered to define a matrix of isotropic, high-resolution voxels in image space. The present claimed invention solves for the unknown high-resolution voxel values to form the isotropic, high-resolution, three-dimensional image in the image space. As discussed during the interview, steps (a) and (b) concern raw data, while step (c) concerns the transition from raw data space to image space and step (d) occurs within the image space.

The projection technique of *Higashi* does not teach or suggest a technique for actually solving for the unknown high-resolution voxel values to form the image in the image space in the manner defined in the amended claims. The remaining references of record do not overcome that deficiency of *Higashi*. Therefore, the Applicants respectfully submit that the invention as presently claimed is patentable over the prior art of record.

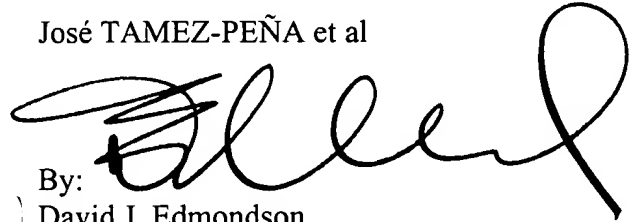
For the reasons set forth above, the Applicants respectfully submit that the application as amended is in condition for allowance. Notice of such allowance is earnestly solicited.

If any questions remain that can be addressed through a telephone communication, the Examiner is invited to telephone the undersigned at the telephone number set forth below.

Please charge any shortage of fees, or credit any overpayment thereof, to BLANK ROME LLP, Deposit Account No. 23-2185 (116741-00138). In the event that a petition for an extension of time is required to be submitted herewith and in the event that a separate petition does not accompany this Response or is insufficient to render this Response timely, the Applicants hereby petition under 37 C.F.R. §1.136(a) for an extension of time for as many months as are required to render this submission timely. Any fee due is authorized above.

Respectfully submitted,

José TAMEZ-PEÑA et al

A large, stylized handwritten signature in black ink, likely belonging to David J. Edmondson, is written over the printed name.

By:
David J. Edmondson
Reg. No. 35,126

Blank Rome LLP
The Watergate Building
600 New Hampshire Avenue, N.W.
Washington, D.C. 20037
202-772-5800 (receptionist)
202-772-5838 (direct dial to attorney signing paper)
202-572-1438 (facsimile)

**RECEIVED****OCT 08 2004****TECH CENTER 2800**

Artificial Intelligence

Second Edition

Elaine Rich

*Microelectronics and Computer
Technology Corporation*

Kevin Knight

Carnegie Mellon University

McGraw-Hill, Inc.
New York St. Louis San Francisco Auckland Bogotá Caracas
Hamburg Lisbon London Madrid Mexico Milan Montreal
New Delhi Paris San Juan São Paulo Singapore Sydney
Tokyo Toronto

Artificial Intelligence

Copyright © 1991, 1983 by McGraw-Hill, Inc.

All rights reserved.

Printed in the United States of America.

Except as permitted under the United States Copyright Act of 1976, no part of this publication may be reproduced or distributed in any form or by any means, or stored in a data base or retrieval system, without the prior written permission of the publisher.

1 2 3 4 5 6 7 8 9 0 DOC DOC 9 5 4 3 2 1 0

ISBN 0-07-052263-4

The editors were David M. Shapiro and Joseph F. Murphy;
the production supervisor was Louise Karam.
R. R. Donnelley & Sons Company was printer and binder.

Library of Congress Cataloging in Publication Data

Rich, Elaine

Artificial Intelligence / Elaine Rich, Kevin Knight.—2nd ed.

p. cm.

Includes bibliographical references and index.

ISBN 0-07-052263-4

I. Artificial intelligence. I. Knight, Kevin. II. Title.

Q335.R53 1991

006.3—dc20

90-20608

ELEC COMPUTER ENG → 912025721438

About the Authors

Elaine Rich is Director of the Artificial Intelligence Laboratory at the Microelectronic and Computer Technology Corporation (MCC). Formerly on the faculty in computer sciences at the University of Texas, she received her Ph.D. from Carnegie Mellon University. Her research interests include natural language processing, knowledge representation, and machine translation.

Kevin Knight received his B.A. from Harvard University and is presently completing his Ph.D. in computer science at Carnegie Mellon University. A regular consultant at MCC, his research interests include natural language processing, unification, machine translation, and search.

BEST AVAILABLE COPY

09/30/04

11:00

3.1 Generate-and-Test

The generate-and-test strategy is the simplest of all the approaches we discuss. It consists of the following steps:

Algorithm: Generate-and-Test

1. Generate a possible solution. For some problems, this means generating a particular point in the problem space. For others, it means generating a path from a start state.
2. Test to see if this is actually a solution by comparing the chosen point or the endpoint of the chosen path to the set of acceptable goal states.
3. If a solution has been found, quit. Otherwise, return to step 1.

If the generation of possible solutions is done systematically, then this procedure will find a solution eventually, if one exists. Unfortunately, if the problem space is very large, "eventually" may be a very long time.

The generate-and-test algorithm is a depth-first search procedure since complete solutions must be generated before they can be tested. In its most systematic form, it is simply an exhaustive search of the problem space. Generate-and-test can, of course, also operate by generating solutions randomly, but then there is no guarantee that a solution will ever be found. In this form, it is also known as the British Museum algorithm, a reference to a method for finding an object in the British Museum by wandering randomly.¹ Between these two extremes lies a practical middle ground in which the search process proceeds systematically, but some paths are not considered because they seem unlikely to lead to a solution. This evaluation is performed by a heuristic function, as described in Section 2.2.2.

The most straightforward way to implement systematic generate-and-test is as a depth-first search tree with backtracking. If some intermediate states are likely to appear often in the tree, however, it may be better to modify that procedure, as described above, to traverse a graph rather than a tree.

For simple problems, exhaustive generate-and-test is often a reasonable technique. For example, consider the puzzle that consists of four six-sided cubes, with each side of each cube painted one of four colors. A solution to the puzzle consists of an arrangement of the cubes in a row such that on all four sides of the row one block face of each color is showing. This problem can be solved by a person (who is a much slower processor for this sort of thing than even a very cheap computer) in several minutes by systematically and exhaustively trying all possibilities. It can be solved even more quickly using a heuristic generate-and-test procedure. A quick glance at the four blocks reveals that there are more, say, red faces than there are of other colors. Thus when placing a block with several red faces, it would be a good idea to use as few of them as possible as outside faces. As many of them as possible should be placed to abut the next block. Using this heuristic, many configurations need never be explored and a solution can be found quite quickly.

¹Or, as another story goes, if a sufficient number of monkeys were placed in front of a set of typewriters, eventually one of them would eventually produce all of the works of Shakespeare.

Unfortunately, for problems much harder than this, even heuristic generate-and-test, all by itself, is not a very effective technique. But when combined with other techniques to restrict the space in which to search even further, the technique can be very effective.

For example, one early example of a successful AI program is DENDRAL (Lindsay *et al.*, 1980), which infers the structure of organic compounds using mass spectrogram and nuclear magnetic resonance (NMR) data. It uses a strategy called *plan-generate-test*, in which a planning process that uses constraint-satisfaction techniques (see Section 3.5) creates lists of recommended and contraindicated substructures. The generate-and-test procedure then uses those lists so that it can explore only a fairly limited set of structures. Constrained in this way, the generate-and-test procedure has proved highly effective.

This combination of planning, using one problem-solving method (in this case, constraint satisfaction) with the use of the plan by another problem-solving method, generate-and-test, is an excellent example of the way techniques can be combined to overcome the limitations that each possesses individually. A major weakness of planning is that it often produces somewhat inaccurate solutions since there is no feedback from the world. But by using it only to produce pieces of solutions that will then be exploited in the generate-and-test process, the lack of detailed accuracy becomes unimportant. And, at the same time, the combinatorial problems that arise in simple generate-and-test are avoided by judicious reference to the plans.

3.2 Hill Climbing

Hill climbing is a variant of generate-and-test in which feedback from the test procedure is used to help the generator decide which direction to move in the search space. In a pure generate-and-test procedure, the test function responds with only a yes or no. But if the test function is augmented with a heuristic function² that provides an estimate of how close a given state is to a goal state, the generate procedure can exploit it as shown in the procedure below. This is particularly nice because often the computation of the heuristic function can be done at almost no cost at the same time that the test for a solution is being performed. Hill climbing is often used when a good heuristic function is available for evaluating states but when no other useful knowledge is available. For example, suppose you are in an unfamiliar city without a map and you want to get downtown. You simply aim for the tall buildings. The heuristic function is just distance between the current location and the location of the tall buildings and the desirable states are those in which this distance is minimized.

Recall from Section 2.3.4 that one way to characterize problems is according to their answer to the question, "Is a good solution absolute or relative?" Absolute solutions exist whenever it is possible to recognize a goal state just by examining it. Getting downtown is an example of such a problem. For these problems, hill climbing can terminate whenever a goal state is reached. Only relative solutions exist, however, for maximization (or minimization) problems, such as the traveling salesman problem. If these problems, there is no *a priori* goal state. For problems of this sort, it makes sense to terminate hill climbing when there is no reasonable alternative state to move to.

²What we are calling the heuristic function is sometimes also called the *objective function*, particularly in the literature of mathematical optimization.

3.2.1 Simple Hill Climbing

The simplest way to implement hill climbing is as follows.

Algorithm: Simple Hill Climbing

1. Evaluate the initial state. If it is also a goal state, then return it and quit. Otherwise, continue with the initial state as the current state.
2. Loop until a solution is found or until there are no new operators left to be applied in the current state:
 - (a) Select an operator that has not yet been applied to the current state and apply it to produce a new state.
 - (b) Evaluate the new state.
 - i. If it is a goal state, then return it and quit.
 - ii. If it is not a goal state but it is better than the current state, then make it the current state.
 - iii. If it is not better than the current state, then continue in the loop.

The key difference between this algorithm and the one we gave for generate-and-test is the use of an evaluation function as a way to inject task-specific knowledge into the control process. It is the use of such knowledge that makes this and the other methods discussed in the rest of this chapter *heuristic* search methods, and it is that same knowledge that gives these methods their power to solve some otherwise intractable problems.

Notice that in this algorithm, we have asked the relatively vague question, "Is one state *better* than another?" For the algorithm to work, a precise definition of *better* must be provided. In some cases, it means a higher value of the heuristic function. In others, it means a lower value. It does not matter which, as long as a particular hill-climbing program is consistent in its interpretation.

To see how hill climbing works, let's return to the puzzle of the four colored blocks. To solve the problem, we first need to define a heuristic function that describes how close a particular configuration is to being a solution. One such function is simply the sum of the number of different colors on each of the four sides. A solution to the puzzle will have a value of 16. Next we need to define a set of rules that describe ways of transforming one configuration into another. Actually, one rule will suffice. It says simply pick a block and rotate it 90 degrees, in any direction. Having provided these definitions, the next step is to generate a starting configuration. This can either be done at random or with the aid of the heuristic function described in the last section. Now hill climbing can begin. We generate a new state by selecting a block and rotating it. If the resulting state is better, then we keep it. If not, we return to the previous state and try a different perturbation.

3.2.2 Steepest-Ascent Hill Climbing

3.2. HILL CLIMBING

climbing or gradient search. Notice that this contrasts with the basic method in which the first state that is better than the current state is selected. The algorithm works follows.

Algorithm: Steepest-Ascent Hill Climbing

1. Evaluate the initial state. If it is also a goal state, then return it and quit. Otherwise, continue with the initial state as the current state.
2. Loop until a solution is found or until a complete iteration produces no change current state:
 - (a) Let *SUCC* be a state such that any possible successor of the current state will be better than *SUCC*.
 - (b) For each operator that applies to the current state do:
 - i. Apply the operator and generate a new state.
 - ii. Evaluate the new state. If it is a goal state, then return it and quit. not, compare it to *SUCC*. If it is better, then set *SUCC* to this state. if it is not better, leave *SUCC* alone.
 - (c) If the *SUCC* is better than current state, then set current state to *SUCC*.

To apply steepest-ascent hill climbing to the colored blocks problem, we must consider all perturbations of the initial state and choose the best. For this problem, it is difficult since there are so many possible moves. There is a trade-off between the time required to select a move (usually longer for steepest-ascent hill climbing) and the number of moves required to get to a solution (usually longer for basic hill climbing that must be considered when deciding which method will work better for a particular problem.

Both basic and steepest-ascent hill climbing may fail to find a solution. Either algorithm may terminate not by finding a goal state but by getting to a state from which no better states can be generated. This will happen if the program has reached either local maximum, a plateau, or a ridge.

A *local maximum* is a state that is better than all its neighbors but is not better than some other states farther away. At a local maximum, all moves appear to make things worse. Local maxima are particularly frustrating because they often occur almost within sight of a solution. In this case, they are called *foothills*.

A *plateau* is a flat area of the search space in which a whole set of neighboring states have the same value. On a plateau, it is not possible to determine the best direction in which to move by making local comparisons.

A *ridge* is a special kind of local maximum. It is an area of the search space that is higher than surrounding areas and that itself has a slope (which one would like to climb). But the orientation of the high region, compared to

There are some ways of dealing with these problems, although these methods are by no means guaranteed:

- Backtrack to some earlier node and try going in a different direction. This is particularly reasonable if at that node there was another direction that looked as promising or almost as promising as the one that was chosen earlier. To implement this strategy, maintain a list of paths almost taken and go back to one of them if the path that was taken leads to a dead end. This is a fairly good way of dealing with local maxima.
- Make a big jump in some direction to try to get to a new section of the search space. This is a particularly good way of dealing with plateaus. If the only rules available describe single small steps, apply them several times in the same direction.
- Apply two or more rules before doing the test. This corresponds to moving in several directions at once. This is a particularly good strategy for dealing with ridges.

Even with these first-aid measures, hill climbing is not always very effective. It is particularly unsuited to problems where the value of the heuristic function drops off suddenly as you move away from a solution. This is often the case whenever any sort of threshold effect is present. Hill climbing is a local method, by which we mean that it decides what to do next by looking only at the "immediate" consequences of its choice rather than by exhaustively exploring all the consequences. It shares with other local methods, such as the nearest neighbor heuristic described in Section 2.2.2, the advantage of being less combinatorially explosive than comparable global methods. But it also shares with other local methods a lack of a guarantee that it will be effective. Although it is true that the hill-climbing procedure itself looks only one move ahead and not at farther, that examination may in fact exploit an arbitrary amount of global information if that information is encoded in the heuristic function. Consider the blocks world problem shown in Figure 3.1. Assume the same operators (i.e., pick up one block and put it on the table; pick up one block and put it on another one) that were used in Section 2.3.1. Suppose we use the following heuristic function:

Local: Add one point for every block that is resting on the thing it is supposed to be resting on. Subtract one point for every block that is sitting on the wrong thing.

Using this function, the goal state has a score of 8. The initial state has a score of 4 (since it gets one point added for blocks C, D, E, F, G, and H and one point subtracted for blocks A and B). There is only one move from the initial state, namely to move block A to the table. That produces a state with a score of 6 (since now A's position causes a point to be added rather than subtracted). The hill-climbing procedure will accept that move. From the new state, there are three possible moves, leading to the three states shown in Figure 3.2. These states have the scores: (a) 4, (b) 4, and (c) 4. The process

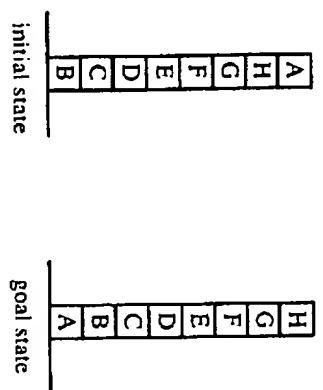


Figure 3.1: A Hill-Climbing Problem

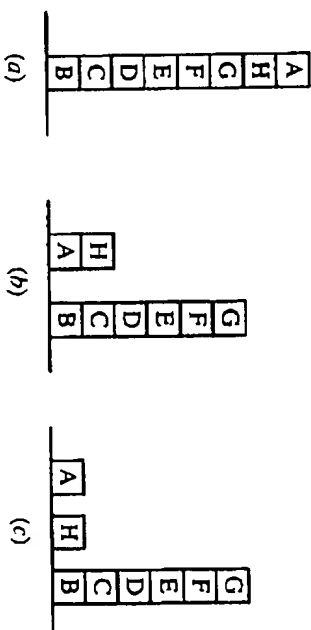


Figure 3.2: Three Possible Moves

than any of its successors because more blocks rest on the correct objects. To solve the problem, it is necessary to disassemble a good local structure (the stack B through I because it is in the wrong global context).

We could blame hill climbing itself for this failure to look far enough ahead to fit a solution. But we could also blame the heuristic function and try to modify it. Suppose we try the following heuristic function in place of the first one:

(Global: For each block that has the correct support structure (i.e., the complete structure underneath it is exactly as it should be), add one point for every block in the support structure. For each block that has an incorrect support structure, subtract one point for every block in the existing support structure.

Using this function, the goal state has the score 28 (1 for B, 7 for C, etc.). The initial state

longer has seven wrong blocks under it. The three states that can be produced next now have the following scores: (a) -28 , (b) -16 , and (c) -15 . This time, steepest-ascent hill climbing will choose move (c), which is the correct one. This new heuristic function captures the two key aspects of this problem: incorrect structures are bad and should be taken apart, and correct structures are good and should be built up. As a result, the same hill climbing procedure that failed with the earlier heuristic function now works perfectly.

Unfortunately, it is not always possible to construct such a perfect heuristic function. For example, consider again the problem of driving downtown. The perfect heuristic function would need to have knowledge about one-way and dead-end streets, which, in the case of a strange city, is not always available. And even if perfect knowledge is, in principle, available, it may not be computationally tractable to use. As an extreme example, imagine a heuristic function that computes a value for a state by invoking its own problem-solving procedure to look ahead from the state it is given to find a solution. It then knows the exact cost of finding that solution and can return that cost as its value. A heuristic function that does this converts the local hill-climbing procedure into a global method by embedding a global method within it. But now the computational advantages of a local method have been lost. Thus it is still true that hill climbing can be very inefficient in a large, rough problem space. But it is often useful when combined with other methods that get it started in the right general neighborhood.

3.2.3 Simulated Annealing

Simulated annealing is a variation of hill climbing in which, at the beginning of the process, some downhill moves may be made. The idea is to do enough exploration of the whole space early on so that the final solution is relatively insensitive to the starting state. This should lower the chances of getting caught at a local maximum, a plateau, or a ridge.

In order to be compatible with standard usage in discussions of simulated annealing, we make two notational changes for the duration of this section. We use the term *objective function* in place of the term *heuristic function*.

And we attempt to *minimize* rather than maximize the value of the objective function.

Thus we actually describe a process of valley descending rather than hill climbing. Simulated annealing [Kirkpatrick *et al.*, 1983] as a computational process is patterned after the physical process of *unraveling*, in which physical substances such as metals are melted (i.e., raised to high energy levels) and then gradually cooled until some solid state is reached. The goal of this process is to produce a minimal-energy final state. Thus this process is one of valley descending in which the objective function is the energy level. Physical substances usually move from higher energy configurations to lower ones, so the valley descending occurs naturally. But there is some probability that a transition to a higher energy state will occur. This probability is given by the function

$$p = e^{-\Delta E/kT}$$

where ΔE is the positive change in the energy level, T is the temperature, and k is Boltzmann's constant. Thus, in the physical valley descending that occurs during

one. Also, the probability that an uphill move will be made decreases as the temperature decreases. Thus such moves are more likely during the beginning of the process when temperature is high, and they become less likely at the end as the temperature becomes lower. One way to characterize this process is that downhill moves are allowed anytime. Large upward moves may occur early on, but as the process progresses, only relative small upward moves are allowed until finally the process converges to a local minimum configuration.

The rate at which the system is cooled is called the *annealing schedule*. Physical annealing processes are very sensitive to the annealing schedule. If cooling occurs too rapidly, stable regions of high energy will form. In other words, a local but not global minimum is reached. If, however, a slower schedule is used, a uniform crystalline structure, which corresponds to a global minimum, is more likely to develop. But, if the schedule is too slow, time is wasted. At high temperatures, where essentially random motion is allowed, nothing useful happens. At low temperatures a lot of time may be wasted after the final structure has already been formed. The optimal annealing schedule for each particular annealing problem must usually be discovered empirically.

These properties of physical annealing can be used to define an analogous process of simulated annealing, which can be used (although not always effectively) whenever simple hill climbing can be used. In this analogous process, ΔE is generalized so that it represents not specifically the change in energy but more generally, the change in the value of the objective function, whatever it is. The analogy for kT is slightly less straightforward. In the physical process, temperature is a well-defined notion, measured in standard units. The variable k describes the correspondence between the units of temperature and the units of energy. Since, in the analogous process, the units of both E and T are artificial, it makes sense to incorporate k into T , selecting values of T that produce desirable behavior on the part of the algorithm. Thus we use the revised probability formula

$$p' = e^{-\Delta E/T}$$

But we still need to choose a schedule of values for T (which we still call temperature). We discuss this briefly below after we present the simulated annealing algorithm.

The algorithm for simulated annealing is only slightly different from the simple hill-climbing procedure. The three differences are:

- The annealing schedule must be maintained.
- Moves to worse states may be accepted.
- It is a good idea to maintain, in addition to the current state, the best state found so far. Then, if the final state is worse than that earlier state (because of bad luck in accepting moves to worse states), the earlier state is still available.

Algorithm: Simulated Annealing

1. Evaluate the initial state. If it is also a goal state, then return it and quit. Otherwise continue with the initial state as the current state.
2. Initialize *BEST-STATE* to the current state.